```
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
  ! Bad coding example 1
  ! !
  ! Shamefully written by Ross Walker (SDSC, 2006)
  !
  ! This code reads a series of coordinates and charges from the file
  ! specified as argument $1 on the command line.
  !
  ! This file should have the format:
  !  I9
  ! 4F10.4   (repeated I9 times representing x,y,z,q)
  !
  ! It then calculates the following fictional function:
  !
  !             exp(rij*qi)*exp(rij*qj)   1
  !    E = Sum( ----------------------- - - )  (rij <= cut)
  !        j<i           r(ij)            a
  !
  ! where cut is a cut off value specified on the command line ($2),
  ! r(ij) is a function of the coordinates read in for each atom and
  ! a is a constant.
  !
  ! The code prints out the number of atoms, the cut off, total number of
  ! atom pairs which were less than or equal to the distance cutoff, the
  ! value of E, the time take to generate the coordinates and the time
  ! taken to perform the calculation of E.
  !
  ! All calculations are done in double precision.
  !+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
double **alloc_2D_double(int nrows, int ncolumns);
void double_2D_array_free(double **array);

/* struct coord
 *
 * This struct is aimed at reducing cache misses during execution.
 * a, b, and c correspond to coord[0], coord[1], and coord[2], respectively.
 */
typedef struct coord_t {
        double a, b, c;
} coord;

int main(int argc, char *argv[])
{
        long natom, i, j;
        long cut_count;

        /* Timer variables */
        clock_t time0, time1, time2;

        double cut;     /* Cut off for Rij in distance units */
        coord *coords; // -> changed to a 1D array of coord structs
        double *q;
        double total_e, current_e, vec2, rij;
        double a;
        FILE *fptr;
```

```c
        char *cptr;

        a = 3.2;

        time0 = clock(); /*Start Time*/
        printf("Value of system clock at start = %ld\n",time0);

        /* Step 1 - obtain the filename of the coord file and the value of
           cut from the command line.
           Argument 1 should be the filename of the coord file (char).
           Argument 2 should be the cut off (float). */
        /* Quit therefore if iarg does not equal 3 = executable name,
           filename, cut off */
        if (argc != 3)
        {
                printf("ERROR: only %d command line options detected", argc-1);
                printf (" - need 2 options, filename and cutoff.\n");
                exit(1);
        }
        printf("Coordinates will be read from file: %s\n",argv[1]);

        /* Step 2 - Open the coordinate file and read the first line to
           obtain the number of atoms */
        if ((fptr=fopen(argv[1],"r"))==NULL)
        {
                printf("ERROR: Could not open file called %s\n",argv[1]);
                exit(1);
        }
        else
        {
                fscanf(fptr, "%ld", &natom);
        }

        printf("Natom = %ld\n", natom);

        cut = strtod(argv[2],&cptr);
        printf("cut = %10.4f\n", cut);

        /* Step 3 - Allocate the arrays to store the coordinate and charge
           data */
        // now allocate array of structs
        coords = (coord*)malloc(sizeof(*coords)*natom);
        if ( coords==NULL )
        {
                printf("Allocation error coords");
                exit(1);
        }
        q=(double *)malloc(natom*sizeof(double));
        if ( q == NULL )
        {
                printf("Allocation error q");
                exit(1);
        }

        /* Step 4 - read the coordinates and charges. */
        for (i = 0; i<natom; ++i)
        {
                // we have to read into the 1d array, now
                fscanf(fptr, "%lf %lf %lf %lf",
                                &(coords[i].a), &(coords[i].b),
                                &(coords[i].c), &q[i]);
        }

        time1 = clock(); /*time after file read*/
        printf("Value of system clock after coord read = %ld\n",time1);


        /* Step 5 - calculate the number of pairs and E. - this is the
           majority of the work. */
        total_e = 0.0;
        cut_count = 0;

        // coordiX correspond to coords[X][i] -> coords[i].X
        double coordia;
        double coordib;
        double coordic;

        // this corresponds to q[i]
        double q_i;

        // this is the square of the cutoff to compare with vec2
        double cut2 = cut * cut;

        for (i = 0; i < natom; ++i)
        {
                // load derefernces here
                coordia = coords[i].a;
                coordib = coords[i].b;
                coordic = coords[i].c;

                q_i = q[i];
                for (j = 0; j < i; ++j)
                {
                        // now we use a literal square with new dereferences
                        vec2 = (coordia-coords[j].a)*(coordia-coords[j].a)
                                +(coordib-coords[j].b)*(coordib-coords[j].b)
                                +(coordic-coords[j].c)*(coordic-coords[j].c);
                        /* X^2 + Y^2 + Z^2 */
                        /* Check if this is below the cut off */

                        // we moved the sqrt inside the if then action and
                        // now compare to cut^2
                        if ( vec2 <= cut2 )
                        {
                                rij = sqrt(vec2); // <- moved here
                                /* Increment the counter of pairs below cutoff */
                                ++cut_count;
                                // now we add the multiples of the exponents in one
                                // exp usage
                                current_e = exp(rij*(q_i+q[j]))/rij;
                                // moved - 1.0/a; until after all the for loops
                                total_e = total_e + current_e;
                        }
                } /* for j=1 j<=natom */
        } /* for i=1 i<=natom */

        // moved here, fixed for not being calculated cut_count times
        total_e -= cut_count / a;

        time2 = clock(); /* time after reading of file and calculation */
        printf("Value of system clock after coord read and E calc = %ld\n",
                        time2);

        /* Step 6 - write out the results */
        printf("                         Final Results\n");
        printf("                         -------------\n");
        printf("                   Num Pairs = %ld\n",cut_count);
        printf("                     Total E = %14.10f\n",total_e);
        printf("     Time to read coord file = %14.4f Seconds\n",
                        ((double )(time1-time0))/(double )CLOCKS_PER_SEC);
        printf("         Time to calculate E = %14.4f Seconds\n",
                        ((double )(time2-time1))/(double )CLOCKS_PER_SEC);
        printf("        Total Execution Time = %14.4f Seconds\n",
                        ((double )(time2-time0))/(double )CLOCKS_PER_SEC);

        /* Step 7 - Deallocate the arrays - we should strictly check the
           return values here but for the purposes of this tutorial we can
           ignore this. */
        free(q);
        //double_2D_array_free(coords);
        // now we just allocate the 1d array like normal
        free(coords);

        fclose(fptr);

        exit(0);
}

double **alloc_2D_double(int nrows, int ncolumns)
{
        /* Allocates a 2d_double_array consisting of a series of pointers
           pointing to each row that are then allocated to be ncolumns
           long each. */

        /* Try's to keep contents contiguous - thus reallocation is
           difficult! */

        /* Returns the pointer **array. Returns NULL on error */
        int i;

        double **array = (double **)malloc(nrows*sizeof(double *));
        if (array==NULL)
                return NULL;
        array[0] = (double *)malloc(nrows*ncolumns*sizeof(double));
        if (array[0]==NULL)
                return NULL;

        for (i = 1; i < nrows; ++i)
                array[i] = array[0] + i * ncolumns;

        return array;

}

void double_2D_array_free(double **array)
{
        /* Frees the memory previously allocated by alloc_2D_double */
        free(array[0]);
        free(array);
}
```

241: